IN-61-CR

43121

p, 34

Advanced Software Development Workstation Project Phase III Final Report

Inference Corporation

February 15, 1991

Cooperative Agreement NCC 9-16
Research Activity No. SE.25

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

ricis

Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

Advanced Software Development Workstation Project Phase III Final Report

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Inference Corporation. Dr. Charles Mckay served as RICIS research coordinator.

Funding has been provided by the Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Robert T. Savely, of the Software Technology Branch, Information Technology Division, Information Systems Directorate, NASA/JSC.

The views and conclusions contained in this report are those of the author and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.

PAGE______INTENTIONALLY BLANK

Advanced Software Development Workstation Project Phase III Final Report

UHCL RICIS Contract NCC-9-16 SE.25

Prepared for
Research Institute for Computing and Information Systems
University of Houston - Clear Lake
and
Software Technology Branch
Information Technology Division
Information Systems Directorate
Johnson Space Center
National Aeronautics and Space Administration

15 February 1991

Prepared by
Inference Corporation
550 N. Continental Blvd.
El Segundo, CA 90245
(213) 322-0200

Table of Contents

1. Introduction **1**
1.1 Motivation 1
1.2 Project Background 1
1.3 Status 1
2. Software Information Systems and Case-based Reasoning **3**
2.1 Background: Reuse-Oriented Software Information Systems 3
2.2 Software reuse as a case-based reasoning process 3
3. The ACCESS Case-based Reasoning Shell **5**
3.1 Knowledge representation 5
3.2 Specification 7
3.3 Retrieval 7
3.4 Modification 8
4. User Interaction with ACCESS **10**
4.1 ACCESS Tools Panel 10
4.2 Browsing or Modifying an Object - the Form Panel 12
4.2.1 The Generic Form Panel 12
4.2.2 Custom Forms 15
4.3 Tools Panel Menus 15
4.3.1 The Object Menu - Saving, Deleting, or Displaying Source Code 16
4.3.2 The File Menu - Saving the Knowledge Base 19
5. Test Sites **20**
5.1 Generation of SVDS Runstreams 20
5.2 Generation of ASDS Drivers 21
5.3 Generation and Execution of Flight Operations Planning and Analysis System Input Processors 22
6. Future Directions **23**
7. Conclusions **24**

List of Figures

Figure 3-1: An associative memory for objects 8
Figure 4-1: ACCESS Tools Panel 11
Figure 4-2: Prompt for Object Name 13
Figure 4-3: ACCESS Generic Form Panel 14
Figure 4-4: Panel with Warning of Constraint Violation 16
Figure 4-5: Example of a Custom Form 17
Figure 4-6: Display of Source Code via Source Panel 18

1. Introduction

1.1 Motivation

Software development is widely considered to be a bottleneck in the development of complex systems, both in terms of development and in terms of maintenance of deployed systems. Cost of software development and maintenance can also be very high. One approach to reducing costs and relieving this bottleneck is increasing the reuse of software designs and software components. A method for achieving such reuse is a software parts composition system. Such a system consists of a language for modeling software parts and their interfaces, a catalog of existing parts, an editor for combining parts, and a code generator that takes a specification and generates code for that application in the target language. The Advanced Software Development Workstation is intended to be an expert system shell designed to provde the capabilities of a software parts composition system.

1.2 Project Background

The first phase of the Automated Software Development Workstation Project began in the fall of 1985 and work has continued to the present. The first phase demonstrated in a limited domain (Space Station momentum management) the feasibility of a knowledge-based approach to the development of a software components composition system. The second phase, which began in April, 1987, focused on ways to exploit knowledge representation, retrieval, and acquisition techniques to reduce the amount of effort required to build such systems. The third phase focused on enhancement of the prototype system developed in Phase II and addressed issues of scale-up and integration with present or future NASA software development environments. In the current extension of the Phase III work, emphasis has been on technology transfer to groups within NASA Johnson Space Center (JSC) and the NASA community and the use of the ASDW prototype in actual software configuration activities. In addition, work has been done on the generalization of the ASDW framework to support use of the system as a generic design knowledge acquisition system.

1.3 Status

ACCESS is the current prototype software for the ASDW. ACCESS is a knowledge-based software information system designed to assist the user in modifying or configuring retrieved software or design objects to satisfy user specifications.

The basis for the representing knowledge in ACCESS is ART-IM®, a toolkit for the development of knowledge-based expert systems. The ART-IM schema system is used as the mechanism for representing objects within ACCESS. ART-IM rules are used to propagate constraints within the object system and to test for constraint violations.

The user interacts with ACCESS via a graphical, point and click interface. This interface has been developed using the facilities of TAE Plus (Transportable Applications Environment Plus), which provides capabilities for developing interfaces on top of the X Window System. The user interface to ACCESS is designed to hide the details of the ART-IM language from the end user. Standard panels are provided for the user to browse and modify objects in the knowledge base. In addition, the knowledge engineer who develops a knowledge base to use with ACCESS can also use TAE Plus develop custom forms for browsing or modification.

The current prototype runs on Sun hardware, but as the source language for ART-IM, TAE Plus, and ACCESS is C, ACCESS is readily portable to a wide variety of platforms.

ACCESS is currently being used at JSC and McDonnell Douglas Corporation to develop domain specific knowledge bases. These development efforts are described briefly in Chapter 5.

2. Software Information Systems and Case-based Reasoning

2.1 Background: Reuse-Oriented Software Information Systems

An important part of the process of developing and maintaining software systems is software reuse, i.e., the activity of retrieving and modifying existing software components, either for the continuing maintenance of an existing system, or for the reuse of components in the development of new software. This reuse-oriented software development activity can be described in the following way (adapted from [18]):

```
begin
  get specification of desired component;
  retrieve best matching component;
  modify component to satisfy specification;
end;
```

Reuse-oriented software processes in the development of software systems have been shown in some experiments to reduce design and maintenance costs [13], [20]. The development of automated tools to support reuse has been a major focus for a government initiative in software productivity [3].

Automated support for software reuse has traditionally been accomplished by on-line catalogs of software components with a user interface based on text information retrieval mechanisms [7], [2]. More recently, work has been conducted towards the construction of *software information systems* [4] that use a knowledge representation language to express a knowledge base describing a software system and its components. Example software information systems are the RLF Librarian [15], AIRS [9], LaSSIE [4], the reuse system developed for abstract data types by Embley and Woodfield [5], and the conceptual information retrieval system for software components developed by Wood and Sommerville [25]. The advantages of a software information system of this type are direct support for semantic retrieval, intelligent indexing, and the use of inheritance and automatic classification for updates [4].

2.2 Software reuse as a case-based reasoning process

To date, work on software information systems has focused on the support of the specification and retrieval parts of the reuse process described above. In the context of the Automated Software Development Workstation project [8], we are exploring the extension of the software information system paradigm through the use of *case-based reasoning* to automate the modification part of the reuse process.

Case-based reasoning is a problem-solving paradigm that adapts stored problem solutions, or cases, to solve new problems specified by a user. It has been applied to a range of classification and construction tasks, and is particularly useful in tasks where a formal set of rules for generating solutions is difficult to obtain, while examples of correct solutions to problems are readily available. The case-based reasoning process can be modeled as follows:

```
begin
  get problem specification;
  retrieve best matching case;
  modify matching case solution to solve problem;
end
```

The case-based reasoning process model as described is clearly an abstraction of the software reuse process model, as has been noted by Williams [24]. The next chapter describes ACCESS, a case-based reasoning shell, in more detail.

3. The ACCESS Case-based Reasoning Shell

ACCESS supports the development of knowledge bases that represent requirements and design templates in reusable software libraries (an earlier version of the system is described in [1]). Application developers use ACCESS to enter a partial specification of an object, retrieve the most closely matching objects stored in the knowledge base, and modify the retrieved object into a new object satisfying the initial specification. This chapter describes the knowledge representation, the specification process, the retrieval mechanism, and the modification process in ACCESS in generic terms. Chapter 4 describes these processes as seen by the user though the ACCESS user interface.

3.1 Knowledge representation

ACCESS knowledge bases consist of a set of objects organized in an inheritance hierarchy. Objects are either classes, representing sets of objects with default attribute values, or instances, representing concrete objects. Objects are represented using the schema-based knowledge representation of the ART-IM knowledge base development system [10]. An example class taxonomy, taken from a ACCESS knowledge base describing a ephemeris generation software package [16], is presented below in outline format:

```
OBJECT
  SOLAR_SYSTEM_OBJECT
    STAR
    SATELLITE
    PLANET
  ASTRONOMICAL_DATUM
    TIME_INTERVAL
    COORDINATES
      SPACE_COORDINATES
      TIME_COORDINATES
        DATE
  APPLICATION
    EPHEMERIS
  RULE
    CONSTRAINT
      FORMULA
      TEMPLATE
      PREDICATE
```

An example of an object instance is the following schema, which represents a code fragment in Ada for a data structure representing a date in the ephemeris generation knowledge base:

```
(DEFSCHEMA DEC-31ST-1988
  (INSTANCE-OF DATE)
  (DAY 31)
  (HOURS 0)
  (MILLISECONDS 0)
  (MINUTE)
  (MINUTES 0)
  (MONTH 12)
  (SECONDS 0)
  (TEXT "(1988,12,31,0,0,0,0)")
  (YEAR 1988))
```

The set of all instances in the knowledge base forms the case base.

Modification rules in the knowledge base are also represented as objects. These rules are based on the various types of knowledge represented and used in the propose-and-revise problem-solving architecture of the SALT system [14]. Currently defined rule classes include:

- *Formulas*: rules that calculate an attribute value for an object by applying a function to other attribute values of the object or of its subobjects (i.e., objects that are values of attributes of the given object).

- *Templates*: rules that compute a string-valued attribute value for an object by instantiating a string template with other attribute values of the object or of its subobjects.

- *Predicates*: rules that perform a procedural test on an object. If the test fails, the object is marked as having a constraint violation.

An example of a modification rule is the following template, which was applied to generate the code fragment in the DEC-31ST-1988 object shown above:

```
(DEFSCHEMA DATE-TEMPLATE
  (INSTANCE-OF TEMPLATE)
  (ARGUMENTS (YEAR MONTH DAY HOURS MINUTES SECONDS MILLISECONDS))
  (ATTRIBUTE TEXT)
  (CONSTRAINT_OF DATE)
  (TEMPLATE_STRING "(%a,%a,%a,%a,%a,%a,%a)"))
```

As in SALT, the modification rule objects are automatically transformed at knowledge base initialization time into ART-IM production rules. The execution of applicable modification rules is performed using the ART-IM inference engine, which uses a modified version of the Rete algorithm [6] to determine which rules can be applied to objects in the knowledge base. A justification-based truth maintenance system manages the consistency of the attribute values associated with the objects in the knowledge base, and automatically retracts attribute values asserted by modification rules when support for the values is retracted.

3.2 Specification

The user of ACCESS enters a new specification of a software object through the following procedure:

```
begin
 select default specification from a class in the taxonomy;
 repeat
    retrieve matches for specification;
    manually modify specification using features from matches;
    execute applicable modification rules;
 until done;
end;
```

This process is known as *specification-by-reformulation* [26]. In the ACCESS user interface, this process is supported using the same style of multi-windowed mouse-driven user interface used in the ARGON [17], BACKBORD [26] and LaSSIE [4] systems. As in the BACKBORD system, specification-by-reformulation is also used as a framework for the interactive acquisition of new object classes and instances from knowledge base developers.

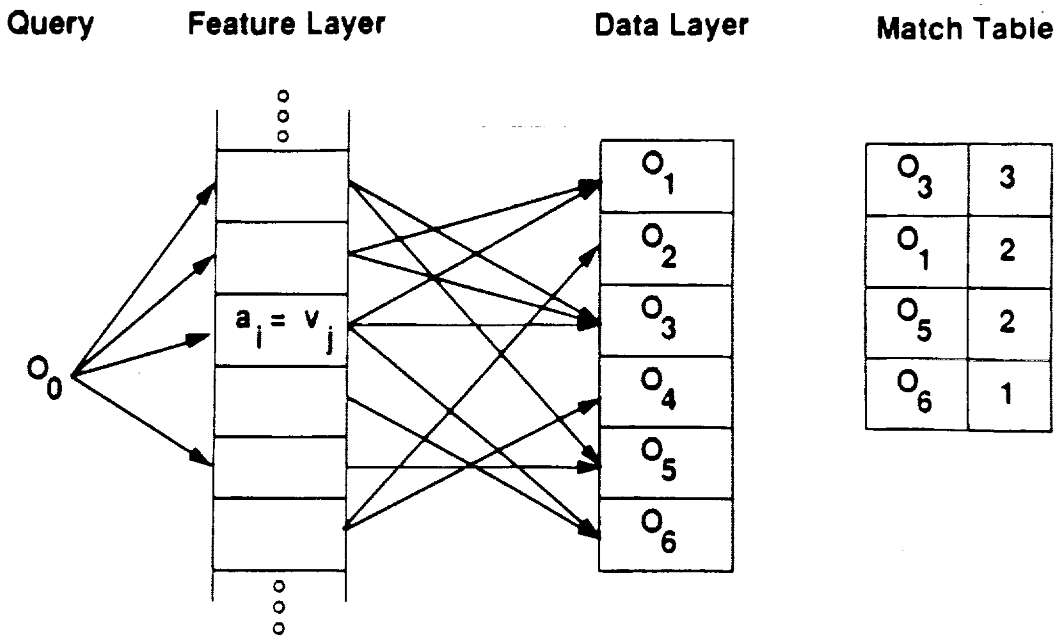
3.3 Retrieval

Given a partial specification of an object as a query, retrieval of objects in the knowledge base is performed in ACCESS using a associative memory implemented using the redundant hash-addressing (RHA) algorithm [23].

An associative memory for object retrieval can be implemented as a two-layered connectionist network, where a *feature layer*, where each unit stands for an attribute/value pair, and a *data layer*, where each unit stands for an instance in the knowledge base [11]. A node in the feature layer is connected to a node in the data layer if and only if the feature represented by the feature layer node is present in the instance represented by the data layer node. In a RHA implementation of such a network, the data layer is an array of pointers to instances, and the feature layer is represented using a hash table with buckets corresponding to each feature containing pointers to sets of array elements. A *match table* is used to accumulate activation levels for units in the data layer during the retrieval process. Figure 3-1 shows the data structures involved in the associative memory.

The associative memory supports the retrieval of the nearest neighbors of an object in a qualitative feature space. This is due to the fact that the distance in the feature space `D(O1,O2)` between two objects `O1` and `O2` is calculated by the following formula, where `A(O)` is the number of features associated with an object and `M(O1,O2)` are the number of matching features between objects `O1` and `O2` [12]:

```
D(O1,O2) = max(A(O1),A(O2)) - M(O1,O2)
```



where O_i is an object, a_j is an attribute, and v_k is a value

Figure 3-1: An associative memory for objects

When a is the maximum number of attributes for an object, and n is the number of objects stored in memory, the complexity of the algorithm is O(a) * O(n) for preprocessing time, O(a) + O (n) for retrieval time, and the storage cost is O(n) + O(a*n) [21]. This compares favorably with the Tree-Hash case retrieval algorithm [22] in situations where the number of attributes associated with cases is large and varies frequently during the process of knowledge acquisition, as is the case in the ACCESS.

Currently, the associative memory exists completely in the virtual memory of ACCESS process. A simple extension of the algorithm to include a hierarchical memory organization (based on that used in commercial text database retrieval systems) can allow the algorithm to handle knowledge bases with 10^5 objects [19], [12].

3.4 Modification

Modification of ACCESS objects results from a combination of manual editing and constraint propagation. Having partially specified an object, the user can edit the best matching object, thereby creating a new object matching the desired specification. Then any applicable constraints embodied in the knowledge base will be propagated in the modified object. For example, suppose the instance of a DATE object shown above, DEC-31-1988, is copied and modified by changing the value of the **day** attribute to 15 and renaming the object DEC-15TH-1988. The modification rule embodied in DATE-TEMPLATE will then result in the modification of the TEXT attribute to reflect the correct code fragment, as indicated below:

```
(DEFSCHEMA DEC-15TH-1988
  (INSTANCE-OF DATE)
  (DAY 15)
  (HOURS 0)
  (MILLISECONDS 0)
  (MINUTE)
  (MINUTES 0)
  (MONTH 12)
  (SECONDS 0)
  (TEXT "(1988,12,15,0,0,0,0)")
  (YEAR 1988))
```

4. User Interaction with ACCESS

The ACCESS user interface provides capabilities for the user to browse the taxonomy of a knowledge base, select a particular instance (case) or class from this knowledge base as the "current object", and browse/edit either the currently selected case or, if the current object is a class, an instantiation of the selected class. Once an object has been selected, a list of objects matching that object is displayed along with a measure of the degree of matching. Any of these objects can then be selected as the current object. Once an object is "opened" for browsing or editing, it can be copied and modified to meet the user's specifications using either a generic form supplied as part of the ACCESS shell or custom forms which have been developed by the knowledge engineer to support a particular knowledge base.

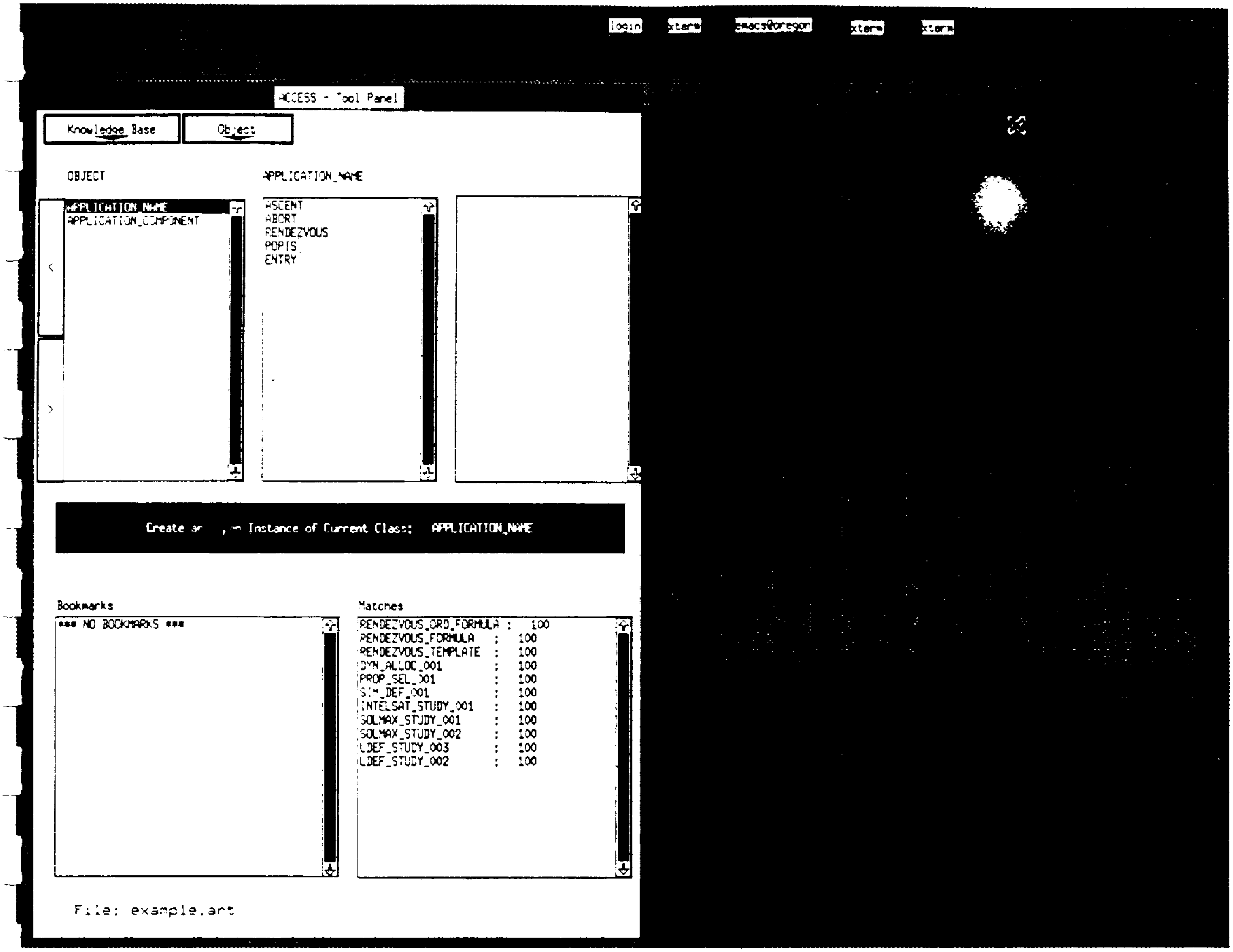
In addition, there are options on an "object menu" to allow the user to compare the current object's features with those of any other object in the same class and to allow the user to browse any source code generated for the object and, optionally, write it to a text file. Other options on this and the "knowledge base" menu allow the user to "save" newly created objects or to "save" the entire knowledge base to a file.

The following sections describe in more detail ACCESS functionality as it appears from the end user's perspective. The examples in this section are taken from a knowledge base being developed to represent an orbital trajectory simulation system, SVDS. In this case, the code generation facilities which are supported by ACCESS are used to build input streams for the simulation program.

4.1 ACCESS Tools Panel

The ACCESS Tools Panel displayed in Figure 4-1 is the first panel displayed upon invocation of ACCESS. The top half of this panel is used to display the taxonomy of the knowledge base. Within this region are three taxonomy subpanels or "windows." If an object in one of these subpanels is selected (by pointing and clicking) with the mouse, it is highlighted and becomes the "current object." If this object has children, then a list of these children is displayed in the next window to the right, with the name of the previously selected object displayed above. If an object is selected from the rightmost taxonomy window, then the taxonomy display is shifted one panel left before displaying the child list. Two buttons on the left of the taxonomy windows offer the user the option of shifting the display of the currently displayed taxonomy either right or left.

In the center there is a subpanel or "button" called the Open Object Button. This subpanel displays the name of the current object - that object which has been most recently selected from the taxonomy windows. If the current object is an object instance (a case), then this button offers the option of "opening" the object for



OBJECT

```
APPLICATION_NAME
APPLICATION_COMPONENT
```

APPLICATION_NAME

```
ASCENT
ABORT
RENDEZVOUS
POPIS
ENTRY
```

Bookmarks

```
*** NO BOOKMARKS ***
```

Matches

```
RENDEZVOUS_ORD_FORMULA :   100
RENDEZVOUS_FORMULA     :  100
RENDEZVOUS_TEMPLATE    :  100
DYN_ALLOC_001          :  100
PROP_SEL_001           :  100
SIM_DEF_001            :  100
INTELSAT_STUDY_001     :  100
SOLMAX_STUDY_001       :  100
SOLMAX_STUDY_002       :  100
LDEF_STUDY_003         :  100
LDEF_STUDY_002         :  100
```

File: example.art

Figure 4-1: ACCESS Tools Panel

browsing or editing. Clicking on this button will invoke either the generic Form Panel or a custom form through which the object can be modified. If the current object is a class, then this button offers the option of creating an instance of that class and then opening the resulting instance for browsing or modification.

In the lower half of the Tools Panel are two windows, the Bookmarks Window and the Matches Window. The Bookmarks Window displays a chronologically ordered list of objects which have been opened during this ACCESS session. The Matches windows provides a list of objects in the same class as the current object, ordered by the extent to which they "match" the current object. Matching between two objects is done by comparing the features of one object with those of the other - each feature which matches increases the level of matching.

When an object is selected from either the Bookmarks Window or Matches Window by a mouse click, it becomes the current object. When this happens, the taxonomy windows are updated to display the ancestors and siblings of this object. The Open Object Button is updated to show the name of the new current object, and the Matches Window is also updated.

At the bottom of the Tools Panel is a display showing the name of the file from which the current knowledge base was loaded or to which it has been saved in the course of the current session.

4.2 Browsing or Modifying an Object - the Form Panel

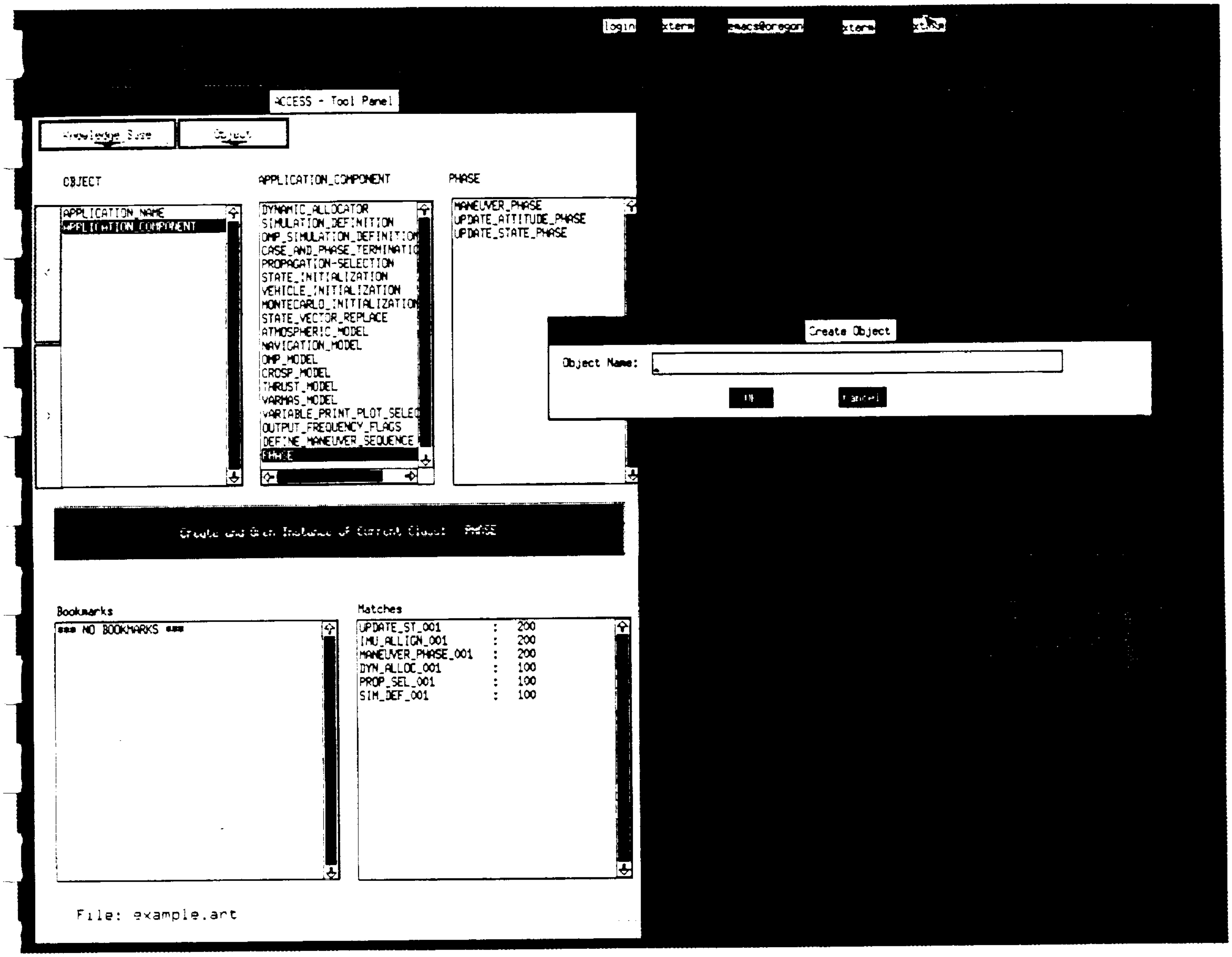
In order to open an object or an instance of a class for browsing and/or editing, the user clicks on the Open Object Button in the center of the Tools Panel. If the current object represents a class, rather than an object instance, the user will be prompted for the name of a new instance, as illustrated in Figure 4-2. If the user supplies such a name, that becomes the name of the new current object.

4.2.1 The Generic Form Panel

Once an object instance is opened, the generic Form Panel will be displayed (unless a custom form has been specified for the class to which the object belongs). When the Form Panel appears, the Tools Panel is "frozen" - that is, it becomes insensitive to mouse clicks and other input. The generic Form Panel is shown in Figure 4-3.

The name of the object being browsed is displayed in the upper left hand corner of the Form Panel. In the top half of the panel is the Object Features subpanel or window. Displayed inside this window are a list of object attributes and values. Attribute names and values are truncated if necessary to conform with the screen size. The user can select a particular feature to examine by pointing and clicking with the mouse.

When a feature is selected, the value corresponding to that feature is displayed in the Attribute Value subpanel, which appears in the lower half of the Form Panel. The user can enter text directly into this subpanel, thus editing the currently selected attribute value. Any such editing must be confirmed by hitting the ESC key. When this is done, the Features display will be updated to show the modified value.

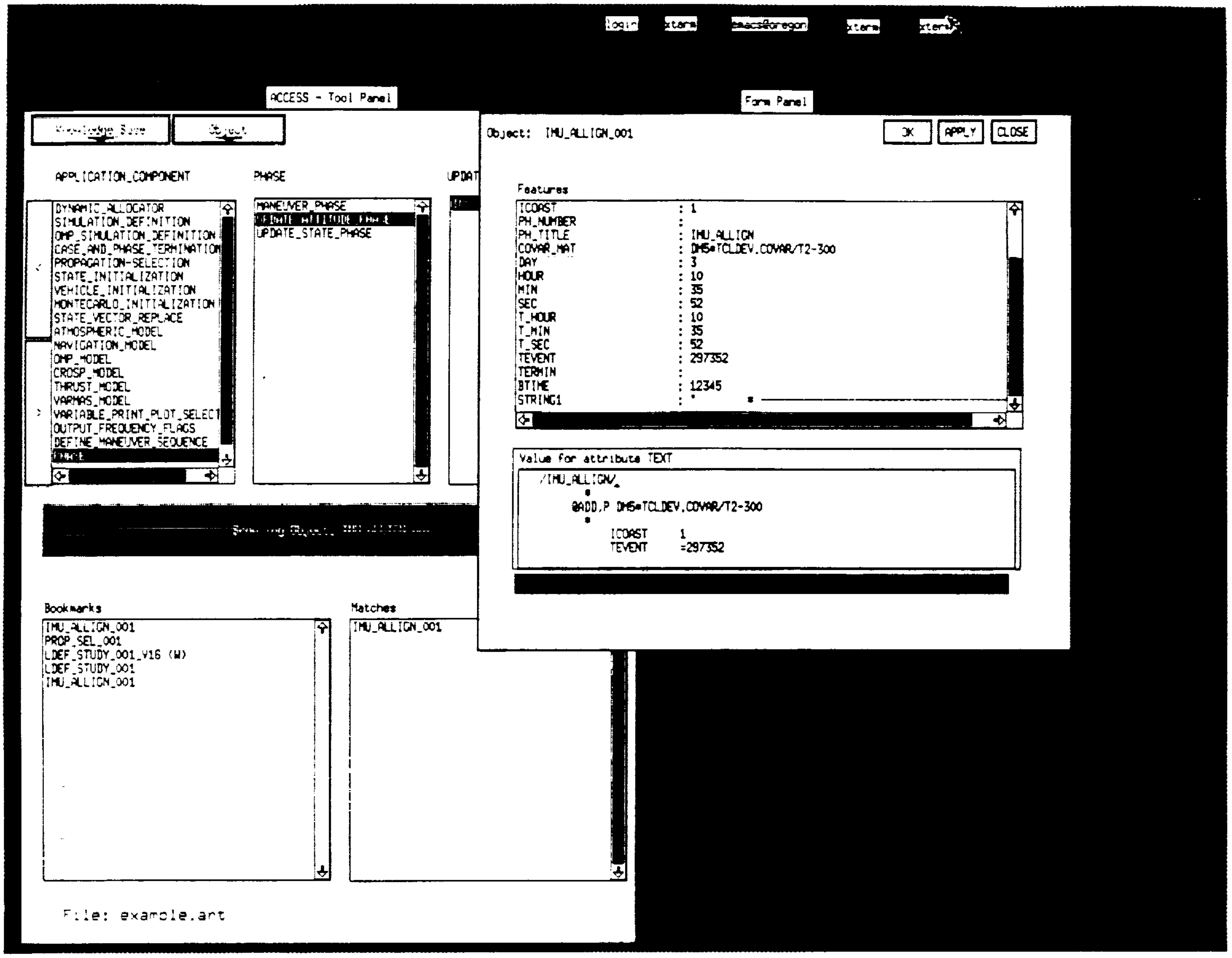


```
OBJECT
APPLICATION_NAME
APPLICATION_COMPONENT

APPLICATION_COMPONENT
DYNAMIC_ALLOCATOR
SIMULATION_DEFINITION
OMP_SIMULATION_DEFINITION
CASE_AND_PHASE_TERMINATIO
PROPAGATION-SELECTION
STATE_INITIALIZATION
VEHICLE_INITIALIZATION
MONTECARLO_INITIALIZATION
STATE_VECTOR_REPLACE
ATMOSPHERIC_MODEL
NAVIGATION_MODEL
OMP_MODEL
CROSP_MODEL
THRUST_MODEL
VARMAS_MODEL
VARIABLE_PRINT_PLOT_SELEC
OUTPUT_FREQUENCY_FLAGS
DEFINE_MANEUVER_SEQUENCE
PHASE

PHASE
MANEUVER_PHASE
UPDATE_ATTITUDE_PHASE
UPDATE_STATE_PHASE

Object Name:

Bookmarks
*** NO BOOKMARKS ***

Matches
UPDATE_ST_001         :   200
IMU_ALLIGN_001        :   200
MANEUVER_PHASE_001    :   200
DYN_ALLOC_001         :   100
PROP_SEL_001          :   100
SIM_DEF_001           :   100

File: example.art
```

Figure 4-2: Prompt for Object Name

Alternatively, if the currently selected attribute is one whose value is restricted to an enumerated set or to an instance of an allowable class of objects, a "SELECT" button will appear to the top and right of this subpanel. By clicking on the SELECT button, the user will cause a menu of allowable values for this attribute to be displayed: he can then select one of them from the menu. If a selection is made, the Features display will be modified to show the newly-selected value.

ORIGINAL PAGE IS
OF POOR QUALITY



```
Object:  IMU_ALLIGN_001

Features
ICOAST       : 1
PH_NUMBER    :
PH_TITLE     : IMU_ALLIGN
COVAR_MAT    : DMS*TCLDEV.COVAR/T2-300
DAY          : 3
HOUR         : 10
MIN          : 35
SEC          : 52
T_HOUR       : 10
T_MIN        : 35
T_SEC        : 52
TEVENT       : 297352
TERMIN       :
BTIME        : 12345
STRING1      : "      "

Value for attribute TEXT
  /IMU_ALLIGN/
       *
     @ADD,P DMS*TCLDEV.COVAR/T2-300
       *
          ICOAST     1
          TEVENT     =297352
```

Figure 4-3: ACCESS Generic Form Panel

Editing within the Attribute window does not change values in objects in the knowledge base until the user clicks on either the "APPLY" or "OK" button in the upper right hand corner of the Form Panel. Clicking on the APPLY button causes the changes which have been recorded on the Form Panel to be made to the object in the knowledge base or, if the object being edited is a "SAVED" object, to a copy of that object.

Within the knowledge base, objects are considered to be either "SAVED" or

"WORKING." SAVED objects are those which were read into the knowledge base at initialization time or which have been explicitly saved by the user (see Subsection 4.3.1). No modifications can be made to a SAVED object. A WORKING object, on the other hand, is one which has been created by the user in the course of the current session and has not been explicitly saved.

If the user is editing a SAVED object when he selects APPLY or OK, he will be prompted for a new object name. ACCESS will make a copy of the saved object, assign it the new name and apply the changes to it.

Once any changes to attribute values have been made, any constraints based on these new values are propagated. If constraint violations are detected, then a pop-up panel with a warning message is displayed, with one warning message for each constraint violation. (Figure 4-4 shows the pop-up warning panel.) The Matches Window is also updated based on the new attribute values.

The final button in the top right hand corner of the Form Panel is the "CLOSE" button. Clicking on the CLOSE button causes the Form Panel to be erased from the screen and resensitizes the Tools Panel.

OK is equivalent to APPLY followed by CLOSE.

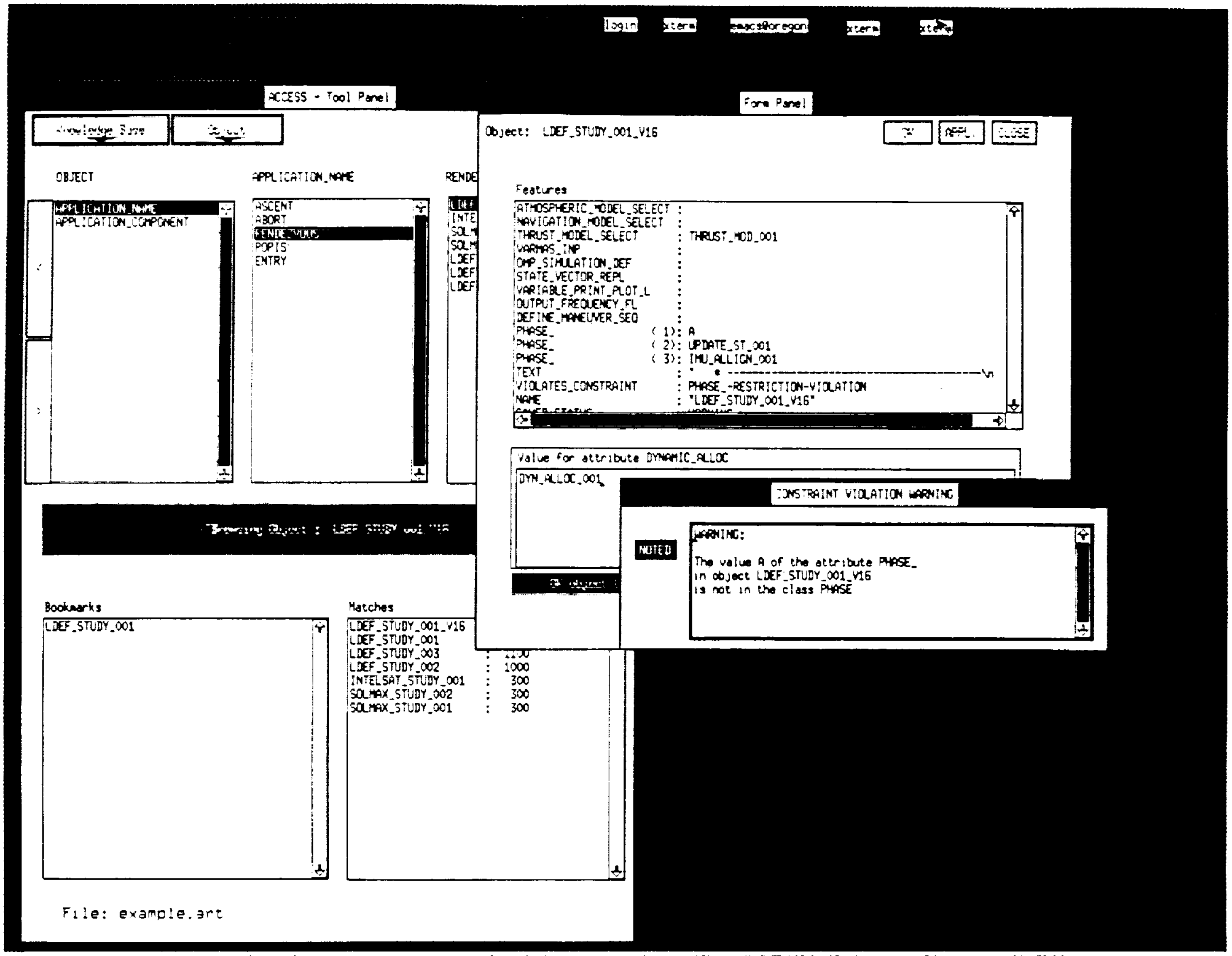
4.2.2 Custom Forms

By default, when an object is browsed or modified, this is done via the generic Form Panel, described above. However, the knowledge engineer who creates a knowledge base can also create custom forms for editing objects which are instances of a particular class. An example of a custom form is given in Figure 4-5. This form is used to specify features of a `propagation-selection` object, which specifies the integration techniques by which an orbit will be propagated. As this example shows, when a custom form is used, various object features can be easily hidden from the user.

Custom forms are specified for an ACCESS application using the screen painting facilities of TAE Plus (Transportable Applications Environment Plus). TAE Plus provides an environment for developing and running window-based applications based on the X Window System. Details on how to specify the interface between TAE forms and ACCESS objects are given in **The ACCESS User's Guide: Building a Knowledge Base.**

4.3 Tools Panel Menus

Near the top of the Tools Panel are two pulldown menus - the Object menu and the Knowledge Base Menu. The functions these menus support are described in the following subsections.



```
OBJECT                      APPLICATION_NAME
APPLICATION_NAME            ASCENT
APPLICATION_COMPONENT       ABORT
                            RENDEZVOUS
                            POPIS
                            ENTRY
```

Browsing Object : LDEF STUDY 001 V16

Bookmarks

```
LDEF_STUDY_001
```

Matches

```
LDEF_STUDY_001_V16
LDEF_STUDY_001
LDEF_STUDY_003
LDEF_STUDY_002       :  1000
INTELSAT_STUDY_001   :   300
SOLMAX_STUDY_002     :   300
SOLMAX_STUDY_001     :   300
```

File: example.art

Object: LDEF_STUDY_001_V16

Features

```
ATMOSPHERIC_MODEL_SELECT   :
NAVIGATION_MODEL_SELECT    :
THRUST_MODEL_SELECT        : THRUST_MOD_001
VARMAS_INP                 :
OMP_SIMULATION_DEF         :
STATE_VECTOR_REPL          :
VARIABLE_PRINT_PLOT_L      :
OUTPUT_FREQUENCY_FL        :
DEFINE_MANEUVER_SEQ        :
PHASE_               ( 1): A
PHASE_               ( 2): UPDATE_ST_001
PHASE_               ( 3): IMU_ALLIGN_001
TEXT                       : "    * ---------------------------------\n
VIOLATES_CONSTRAINT        : PHASE_-RESTRICTION-VIOLATION
NAME                       : "LDEF_STUDY_001_V16"
```

Value for attribute DYNAMIC_ALLOC

```
DYN_ALLOC_001
```

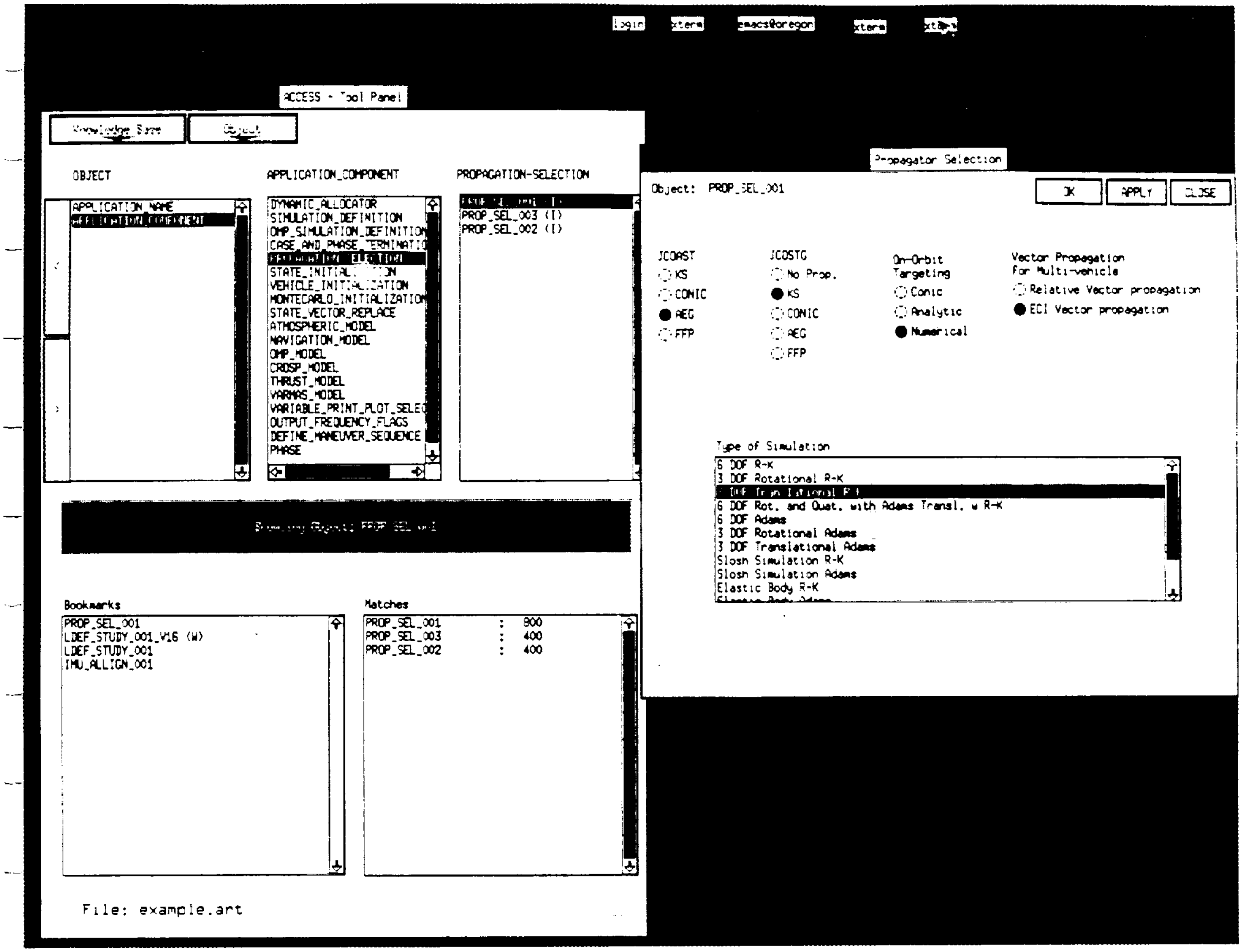
```
WARNING:

The value A of the attribute PHASE_
in object LDEF_STUDY_001_V16
is not in the class PHASE
```

Figure 4-4: Panel with Warning of Constraint Violation

4.3.1 The Object Menu - Saving, Deleting, or Displaying Source Code

The Object Menu consists of three options - **save**, **delete**, **compare**, and **view source**. Each performs its function on the current object, that is, the object whose name is displayed on the Open Object button.

The **save** option makes the current object a SAVED object. This means that this



OBJECT

APPLICATION_NAME
APPLICATION_COMPONENT

APPLICATION_COMPONENT

DYNAMIC_ALLOCATOR
SIMULATION_DEFINITION
OMP_SIMULATION_DEFINITION
CASE_AND_PHASE_TERMINATIO
PROPAGATION_SELECTION
STATE_INITIALIZATION
VEHICLE_INITIALIZATION
MONTECARLO_INITIALIZATION
STATE_VECTOR_REPLACE
ATMOSPHERIC_MODEL
NAVIGATION_MODEL
OMP_MODEL
CROSP_MODEL
THRUST_MODEL
VARMAS_MODEL
VARIABLE_PRINT_PLOT_SELEC
OUTPUT_FREQUENCY_FLAGS
DEFINE_MANEUVER_SEQUENCE
PHASE

PROPAGATION-SELECTION

PROP_SEL_001 (I)
PROP_SEL_003 (I)
PROP_SEL_002 (I)

Browsing Object: PROP_SEL_001

Bookmarks

PROP_SEL_001
LDEF_STUDY_001_V16 (W)
LDEF_STUDY_001
IMU_ALLIGN_001

Matches

PROP_SEL_001	:	900
PROP_SEL_003	:	400
PROP_SEL_002	:	400

File: example.art

Object: PROP_SEL_001

JCOAST
○ KS
○ CONIC
● AEG
○ FFP

JCOSTG
○ No Prop.
● KS
○ CONIC
○ AEG
○ FFP

On-Orbit Targeting
○ Conic
○ Analytic
● Numerical

Vector Propagation For Multi-vehicle
○ Relative Vector propagation
● ECI Vector propagation

Type of Simulation

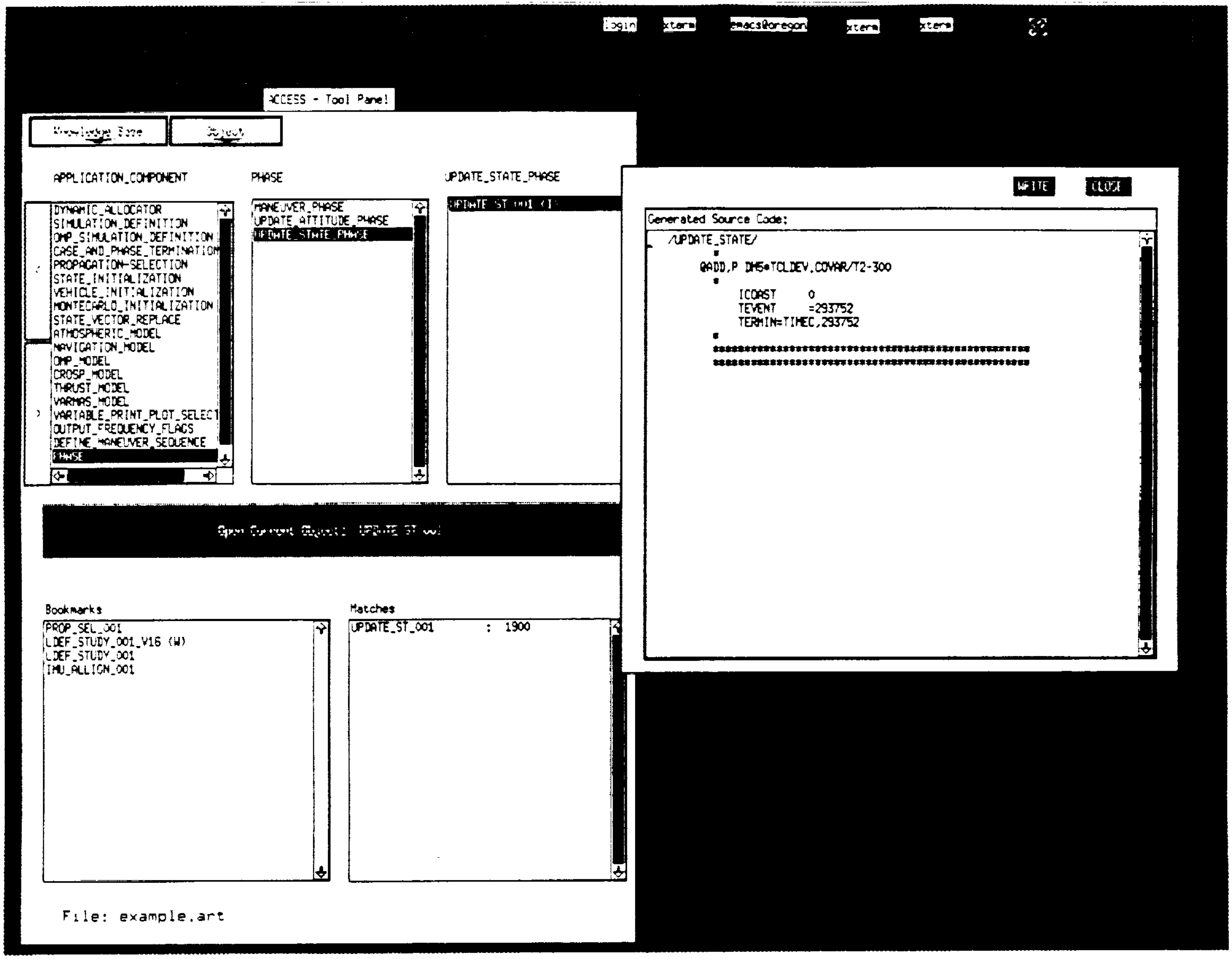
6 DOF R-K
3 DOF Rotational R-K
3 DOF Translational R-K
6 DOF Rot. and Quat. with Adams Transl. w R-K
6 DOF Adams
3 DOF Rotational Adams
3 DOF Translational Adams
Slosh Simulation R-K
Slosh Simulation Adams
Elastic Body R-K
Elastic Body Adams

Figure 4-5: Example of a Custom Form

object can no longer be modified by editing and that its description will be saved if one of the **save** options is selected from the Knowledge Base Menu.

The **delete** option deletes the current object from the knowledge base. If this object appeared on the Bookmarks list, it is deleted from that list. A new object is selected to be the current object and the deleted object will no longer appear in the object taxonomy display.

The **compare** option allows the user to compare the current object with another object in the same class. When this option is selected, a pop-up menu of objects with the same parents is displayed. If the user selects one of these objects, then a panel appears which gives a static display of all features (attribute/value pairs) of the two objects which are different.



APPLICATION_COMPONENT

- DYNAMIC_ALLOCATOR
- SIMULATION_DEFINITION
- OMP_SIMULATION_DEFINITION
- CASE_AND_PHASE_TERMINATION
- PROPAGATION-SELECTION
- STATE_INITIALIZATION
- VEHICLE_INITIALIZATION
- MONTECARLO_INITIALIZATION
- STATE_VECTOR_REPLACE
- ATMOSPHERIC_MODEL
- NAVIGATION_MODEL
- OMP_MODEL
- CROSP_MODEL
- THRUST_MODEL
- VARMAS_MODEL
- VARIABLE_PRINT_PLOT_SELECT
- OUTPUT_FREQUENCY_FLAGS
- DEFINE_MANEUVER_SEQUENCE
- PHASE

PHASE

- MANEUVER_PHASE
- UPDATE_ATTITUDE_PHASE
- UPDATE_STATE_PHASE

UPDATE_STATE_PHASE

- UPDATE_ST_001 (I)

Open Current Object: UPDATE_ST_001

Generated Source Code:

```
 /UPDATE_STATE/
      *
    @ADD,P DMS*TCLDEV.COVAR/T2-300
      *
         ICOAST     0
         TEVENT     =293752
         TERMIN=TIMEC,293752
      *
      ********************************************************
      ********************************************************
```

Bookmarks

- PROP_SEL_001
- LDEF_STUDY_001_V16 (W)
- LDEF_STUDY_001
- IMU_ALLIGN_001

Matches

```
UPDATE_ST_001        :  1900
```

File: example.art

Figure 4-6: Display of Source Code via Source Panel

The **view source** option displays value of the `text` slot of the current object on the

Source Panel. The displayed text can be browsed, but not modified by the user. This option is appropriate when ACCESS is being used to generate source code or an input stream for some software system, as it allows the user to examine the generated code. The Source Panel is shown in Figure 4-6.

At the top of the Source Panel are "WRITE" and "CANCEL" buttons. By selecting the WRITE button, the user causes the text in the **text** slot of the current object (i.e., the text which is displayed on the Source Panel) to be written to a file. The base name for this file is the name of the current object; its suffix is `txt`.

Selecting the CANCEL button returns control to the Tools Panel.

4.3.2 The File Menu - Saving the Knowledge Base

The Knowledge Base Menu consists of three options - **save, save as..**, and **exit.**

When the option **save** is selected, all SAVED objects in the knowledge base as well as ancillary customization data will be written out to the "current" file. This is the file whose name is displayed at the bottom of the Tools Panel - initially, it is the file from which the knowledge base was loaded. If there are WORKING objects in the knowledge base, a warning panel will be displayed and the user will have the option of canceling the save. The file created by **save** can be used as input to a subsequent ACCESS session.

The option **save as..** works in the same way as **save**, except that the user is prompted for the name of a file to save to. When supplied, this becomes the current file.

The **exit** option causes the current ACCESS session to be terminated.

5. Test Sites

Knowledge bases for ACCESS have been or are currently being developed at NASA and at McDonnell Douglas Corporation to support software reuse. These projects are described briefly in the following sections.

5.1 Generation of SVDS Runstreams

Space Vehicle Dynamics Simulation (SVDS) is a large (several hundred thousand lines of Fortran code) computer simulation program used by flight planners at NASA JSC to help design the trajectory and flight plan for space shuttle missions. Its simulation capabilities include a variety of trajectory and vehicle dynamics problems. Input for a typical SVDS run consists of more than one thousand data values.

In order to construct the input stream for a simulation, the user will typically search for an old data set (called a "data deck") which approximates a closely as possible the specifications for the current simulation. The user will then modify it as appropriate.

Data decks may contain comments identifying the variables and their values, but generation and maintenance of such comments depends on the motivation of the user. Similarly, related input variables are usually grouped together, but there is no automatic enforcement of this sort of structure. Because of the large number of input variables, training a new user is a lengthy process. It is difficult even for experienced users to set up a deck without making some errors. Since a typical SVDS run requires hours or days to execute, the consequences of errors can be fairly expensive.

ACCESS is currently being used as a tool to provide an intelligent interface for data deck preparation. Data decks, also called runstreams, have been categorized based on the specific function they perform relative to trajectory generation. Each class of runstream has been divided into logical groups that together comprise the inputs necessary to specify a complete runstream. Each class has associated with it certain "features" - e.g., the type of input values that are necessary to define a specific portion of the simulation. The various classes of runstreams and their features are then represented as ACCESS objects (ART-IM schemas). The development and characterization of this object hierarchy is the key to creating the knowledge base.

Custom forms have been built for various runstream components. These custom forms permit easy data entry and or modification of default inputs. Certain computations which previously had to be performed manually have been expressed in the ACCESS system as constraints. For example, the time of specific mission events is generated automatically from input values of day, hour, minutes, and seconds. Currently, such computations are performed using hand calculators and constitute one of the most error-prone steps in the data deck preparation process.

Once the relevant features of an data deck component have been specified, the ACCESS template mechanism is used to generate automatically the appropriate portion of the runstream. This code may include comments. Once the input values for the required components within a class of runstreams have been specified, a syntactically valid runstream is available for execution.

5.2 Generation of ASDS Drivers

The first application of ACCESS was to build drivers for the Ada Simulation Development System, ASDS, developed by McDonnell Douglas Space Systems Company, Engineering Services Division.

The ASDS is an Ada-based system for developing simulation programs. "Philosophically, ASDS assumes that any simulation can be characterized as propagation separated by discrete events. The two basic functions, propagation and discrete event execution, are captured in the ASDS executive routines in a very general way. Consequently, any specific simulation is rather easily set up, since it requires only the instantiation of the generic logic...When a user wants to create a specific application using ASDS, he/she must develop routines that control all of the discrete events, and must develop functions that "trigger" those events. In addition, if integration is desired, an equations of motion routine must be developed. Also, routines that read input must be developed. Then, a driver can be constructed that instantiates the generic routines with the specific routines developed, and a control loop constructed."[1]

The ACCESS knowledge base for the ASDS includes previously written drivers and driver templates. These are available to the user as a starting point for development of a new driver. Initially, the user makes use of the ACCESS retrieval mechanism to select an existing driver that most closely represents his/her present task. As the user selects routines to be used in the driver, the corresponding Ada code for the driver is constructed automatically. If the user modifies an ACCESS driver component, the corresponding Ada code is updated automatically, providing single point maintenance of the ASDS drivers.

This prototype application of the ACCESS system was demonstrated the first quarter of 1990 in conjunction with McDonnell Douglas' presentation of the Knowledge Based Executive.

[1] Extracted from "The ASDS Executive" by Dr. Robert G. Gottlieb, McDonnell Douglas Space Systems Company, Engineering Services Division.

5.3 Generation and Execution of Flight Operations Planning and Analysis System Input Processors

Another McDonnell Douglas Engineering Services Division project is serving as a test site for application of ACCESS. The project is called Operations Planning and Analysis System, Flight Dynamics Planning and Analysis Software. The software generated in this project makes use of a variation of the ASDS. The difference is that this project is focused on simulation development from the user's perspective. The scope of simulations that can be generated is narrowed and a more complete environment for simulation development and execution is provided. The application of ACCESS concentrates on development of input tables for the predefined processors available.

The ACCESS knowledge base includes a library of input tables that have been created for each of the available processors. Also, available is the ability to generate sequence tables, permitting the processors to be executed in sequence with varying input tables corresponding to the processors. The user selects a category of processor input tables or sequence tables, peruses the examples, and modifies the specific table values via a customized form entry for that class of processor.

Extensions to this project include expanding the ACCESS environment to generically handle creation of the Unix shell scripts required to complete the cycle of simulation creation and execution by the user. Also, integration to existing processor input tables presently stored in a binary record format is required.

6. Future Directions

ACCESS in its current state of development represents a working prototype - that is, it is anticipated that with the creation of suitable knowledge bases, ACCESS should provide demonstrable productivity gains for end users. Thus, in the near term, emphasis will be on enhancing the knowledge bases currently under development and evaluating users' responses to the end result. Numerous changes have been made to the ACCESS user interface during the past year on the basis of perceived user needs. However, it is anticipated that further minor changes will have to be made in response to field test results.

Currently, the ACCESS associative memory is memory-resident. An anticipated task within the next year is to provide an Object Management System (OMS) approach to handling objects in the knowledge base. In particular, this means providing a secondary memory implementation of the associative memory algorithm and developing a method for representing objects in ART-IM schema system on an as-needed basis.

Currently, the development of a knowledge base is still primarily a manual process using text-editing software. Another focus in the coming year should be on developing a Knowledge Engineer's Interface to ACCESS. This goal in developing this interface will be to eliminate the need for a text editor when the KE creates a knowledge base. The features of this interface will be specified after feedback from the current knowledge base development efforts being undertaken at NASA and MDAC. However, the following is a list of possible functionality to be provided:

- Ability for the KE to create new classes and attributes for existing classes.
- An interface for adding constraints and doing consistency and error checking on constraint templates.
- A utility to provide some degree of automation in generating the ART-IM specifications for custom forms from TAE .rfg files.

In addition, a mid-term goal is to provide an implementation of the Engineering Scripting Language whose specification is to be provided by another contractor.

Another direction for enhancement of ACCESS functionality is provide an interface to the underlying operating system for closed loop execution of software developed using ACCESS.

7. Conclusions

ACCESS provides a generic capability to develop software information system applications which are explicitly intended to facilitate software reuse. In addition, it provides the capability to retrofit existing large applications with a user-friendly front end for preparation of input streams in a way that will reduce required training time, improve the productivity even of experienced users, and increase accuracy. Current and past work shows that ACCESS will be scalable to much larger object bases.

References

1. Allen, B.P. and Lee, S.D. A Knowledge-Based Environment for the Development of Software Parts Composition Systems. Proceedings of the 11th International Conference on Software Engineering, IEEE, May, 1989.

2. Arnold, S.P. and Stepoway, S.L. The REUSE System: Cataloging and Retreival of Reusable Software. Proceedings of IEEE Spring COMPCON '87, March, 1987.

3. Batz, J.C., Cohen, P.M., Redwine, S.T. and Rice, J.R. "The Application-Specific Task Area". *IEEE Computer 16*, 11 (November 1983).

4. Devanbu, P., Selfridge, P.G., Ballard, B.W. and Brachman, R.J. A Knowledge-Based Software Information System. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, August, 1989.

5. Embley, D.W. and Woodfield, S.N. A Knowledge Structure for Reusing Abstract Data Types. Proceedings of the 9th International Conference on Software Engineering, IEEE, March-April, 1987.

6. Forgy, C.L. OPS5 User's Manual. Tech. Rept. CMU-CS-81-135, Carnegie-Mellon University Computer Science Department, 1981.

7. Frakes, W.B. and Nejmeh, B.A. Software Reuse Through Information Retrieval. Proceedings of IEEE Spring COMPCON '87, March, 1987.

8. Fridge, E.R. III. The Automated Software Development Workstation Project. Proceedings of the Evolutionary Space Station Symposium, NASA, February, 1990.

9. Hendler, J.A., Wong, Y.C., Vinciguerra, A. and Mogilensky, J. AIRS: An AI-Based Ada Reuse Tool. Proceedings of AIDA-87, November, 1987.

10. Inference Corporation. *ART-IM 2.0 Reference Manual.* Inference Corporation, 1989.

11. Kohonen, T. *Self-Organization and Associative Memory.* Springer-Verlag, 1988. 2nd edition.

12. Kohonen, T. *Content-Addressable Memories.* Springer-Verlag, 1988. 2nd edition.

13. Lanergan, R.G. and Grasso, C.A. Software Engineering with Reusable Designs and Code. In *Software Reusability (V. 2: Applications and Experience)*, Biggerstaff, T.J. and Perlis, A.J., Eds., ACM Press, 1989.

14. Marcus, S. SALT: A Knowledge Acquisition Tools for Propose-and-Revise Systems. In *Automating Knowledge Acquisition for Expert Systems*, Marcus, S., Eds., Kluwer Academic, 1988.

15. McDowell, R.C. and Cassell, K.A. The RLF Librarian: A Reusablility Librarian Based On Cooperating Knowledge-Based Systems. Proceedings of Fourth Annual Knowledge-Based Software Assistant Conference, RADC, September, 1989.

16. U.S. Naval Observatory. *Almanac For Computers 1988.* U.S. Government Printing Office, 1988.

17. Patel-Schneider, P.F., Brachman, R.J., and Levesque, H.J. ARGON: Knowledge Representation meets Information Retrieval. Proceedings of the First Conference on Applications of Artificial Intelligence, IEEE, December, 1984.

18. Prieto-Diaz, R. and Freeman, P. "Classifying Software for Reusability". *IEEE Software 4*, 1 (January 1987).

19. Salton, G. *Automatic Text Processing.* Addison-Wesley, 1989.

20. Selby, R.W. Quantitative Studies of Software Reuse. In *Software Reusability (V. 2: Applications and Experience)*, Biggerstaff, T.J. and Perlis, A.J., Eds., ACM Press, 1989.

21. Stone, H.S. "Parallel Querying of Large Databases: A Case Study". *IEEE Computer 20*, 10 (October 1987).

22. Stottler, R.H., Henke, A.L. and King, J.A. Rapid Retrieval Algorithms for Case-Based Reasoning. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, August, 1989.

23. Venta, O. and Kohonen, T. A Content-Addressing Software Method for the Emulation of Neural Networks. Proceedings of the IEEE International Conference on Neural Networks 1988, IEEE, July, 1987.

24. Williams, R.S. Learning to Program by Examining and Modifying Cases. Proceedings of the DARPA Workshop on Case-Based Reasoning, May, 1988.

25. Wood, M. and Sommerville, I. "An Information Retrieval System for Software Components". *ACM SIGIR Forum 22*, 3,4 (Spring/Summer 1988).

26. Yen, J., Neches, R. and DeBellis, M. BACKBORD: Beyond Retrieval by Reformulation. Tech. Rept. ISI/RS-88-202, USC ISI, March, 1988.